Foundations of Artificial Intelligence

CS472/3
Lecture #6

Bart Selman

Slide CS472–1

Today's Lecture

A*, optimality & completeness. Local search.

Readings: R&N, Chapter 4.

Slide CS472–2

Observations on A*

A* is **optimally efficient**: given the information in h,
no other optimal search method can expand fewer nodes.
Non-trivial and quite remarkable!

Note: A* combines information of cost to get to intermediate nodes (like "uniform cost search") with (under) estimate of cost from intermediate node to goal ("greedy search").

Aside: try uniform cost for fig. 4.5. You get "circles".
More nodes!

Slide CS472–3

A*

Optimal (next)

Complete: Unless there are infinitely many nodes with $f(n) < f^\star$ Assume locally finite:
(1) finite branching, (2) every operator costs at least $\delta > 0$.

Complexity: Still exponential because of breadth-first nature. Unless $|h(n) - h^\star| \leq O(log(h^\star(n))$,
with $h^\star$ true cost of getting to goal.

See: IDA* (p. 106 R&N, "iterative deepening for A*".)

Slide CS472–4

Proof of the optimality of A^*

f is monotonically increasing along any path from the root.

Let G be an optimal goal state, with path cost f^*
Let G_2 be a suboptimal goal state, with path cost $g(G_2) > f^*$
n is a leaf node on an optimal path to G

Because h is admissible, we must have

$$f^* \geq f(n).$$

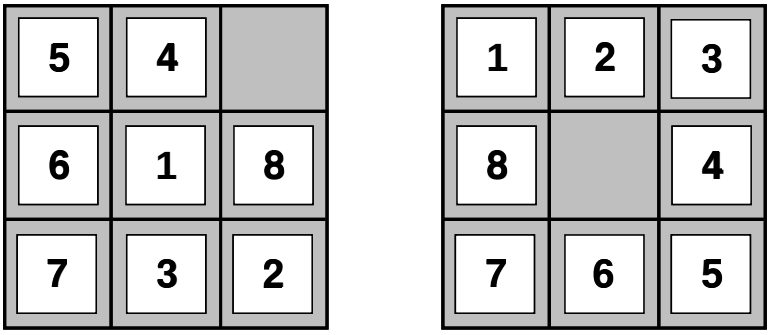
Also, if n is not chosen over G_2, we must have

$$f(n) \geq f(G_2).$$

Gives us $f^* \geq f(G_2) = g(G_2)$. (Then G_2 is *not* suboptimal!)

Slide CS472–5

Heuristic Functions: Example



Slide CS472–6

8-puzzle

1. h_C = number of misplaced tiles
2. h_M = Manhattan distance

Admissible?
Which one should we use?

Slide CS472–7

Importance of $h(n)$

$$h_C \leq h_M \leq h_{\text{opt}}$$

Prefer h_M.
Note: Expand all nodes with $f(n) = g(n) + h(n) < f^\star$
So, $g(n) < f^\star - h(n)$, higher h means fewer n's.
Aside: How would we get an h_{opt} ?

Slide CS472–8

	Search Cost			Effective Branching Factor		
d	IDS	A*(h_1)	A*(h_2)	IDS	A*(h_1)	A*(h_2)
2	10	6	6	2.45	1.79	1.79
4	112	13	12	2.87	1.48	1.45
6	680	20	18	2.73	1.34	1.30
8	6384	39	25	2.80	1.33	1.24
10	47127	93	39	2.79	1.38	1.22
12	364404	227	73	2.78	1.42	1.24
14	3473941	539	113	2.83	1.44	1.23
16	–	1301	211	–	1.45	1.25
18	–	3056	363	–	1.46	1.26
20	–	7276	676	–	1.47	1.27
22	–	18094	1219	–	1.48	1.28
24	–	39135	1641	–	1.48	1.26

Slide CS472–9

Inventing Heuristics

Automatically

A tile can move from sq A to sq B if

A is adjacent to B and B is blank.

(a) A tile can move from sq A to sq B if A is adjacent to B.

(b) A tile can move from sq A to sq B if B is blank.

(c) A tile can move from sq A to sq B.

If all admissible, combine them by taking the *max*.

Slide CS472–10

A Different Approach: Local Search

So far, we have considered methods that systematically explore the full search space, possibly using **principled** pruning (A* etc.).

The current best such algorithms (IDA* / SMA*) can handle search spaces of up to 10^{100} states / around 500 binary valued variables.

(These are "ballpark" figures only!)

Slide CS472–11

What if we have 10,000 or 100,000 variables / search spaces of up to $10^{30,000}$ states?

A completely different kind of method is called for:

Local Search Methods or
Iterative Improvement Methods

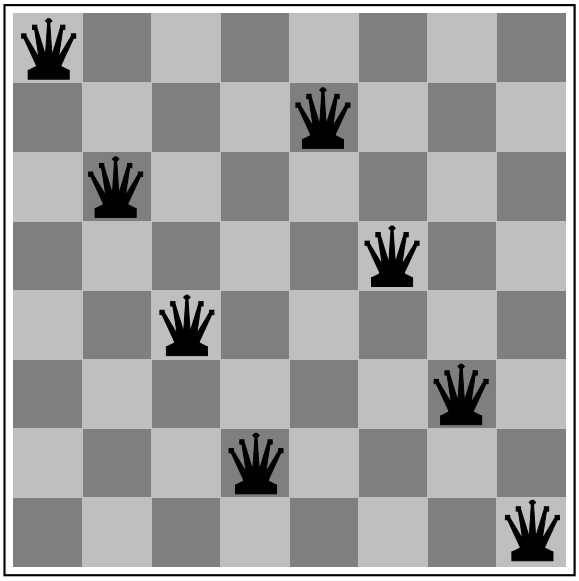
Slide CS472–12

Local Search Methods

Applicable when we're interested in the Goal State — not in how to get there.

E.g. N-Queens, VLSI layout, or map coloring.

Slide CS472–13



Slide CS472–14

Local Search Methods

Approach quite general: any NP-complete problem / CSP.

Other examples: planning, scheduling, TSP, graph coloring, and time-tabling.

In fact, many (most?) large Operations Research problems are solved using local search. E.g. Delta airlines. (near-optimal!)

Q. What's the "competitor" in OR?

Slide CS472–15

Key idea (suprisingly simply):

```
a) Select (random) initial state
     (initial guess at solution)

b) Make local modification to try
        to improve current state.

c) repeat b) till goal state found
        (or out of time).
```

Slide CS472–16

Example

Graph coloring.

a) start with random coloring of nodes

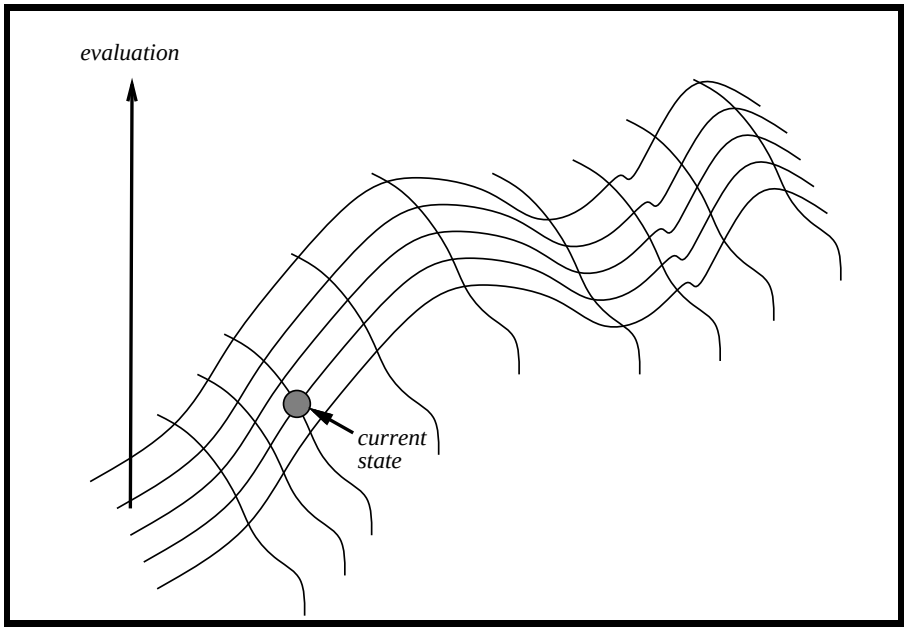
b) change the coloring of one of the nodes to reduce conflicts

c) repeat b)

Slide CS472–17

```
#1  #2   #3  #4  #5 || # conflicts
 R   G    B   R   B ||   1
```

Slide CS472–18



Slide CS472–19